

JP11110256

Publication Title:

DEVICE AND METHOD FOR DEBUGGING PROGRAM, AND COMPUTER READABLE RECORDING MEDIUM RECORDED WITH THE METHOD FOR THE SAME

Abstract:

Abstract of JP 11110256

(A) Translate this text PROBLEM TO BE SOLVED: To debug a program by generating an object file so that the debugging object and the final object are managed with the same execution object a memory of target machine side is not burdened in debugging. SOLUTION: An object file 24 generated by a compiler 21 based on a source file 23 is divided into a debug script file activated in debugging, an execution object file which does not include a debug script, and a symbol table of debug script activation information relating the debug script with a position of a source line. In downloading the execution object file in a target machine 11, executing this file and debugging it, when an execution of the program reaches the debug script part, a debugger 22 executes a debug script file on a host machine on the basis of debug script activation information and executes a debug.

Courtesy of <http://v3.espacenet.com>

(19)日本国特許庁 (J P)

(12) 公 開 特 許 公 報 (A)

(11)特許出願公開番号

特開平11-110256

(43)公開日 平成11年(1999)4月23日

(51)Int.Cl.⁶

G 0 6 F 11/28
9/45

識別記号

3 0 5

F I

G 0 6 F 11/28
9/44

3 0 5 Z
3 2 2 Z

審査請求 未請求 請求項の数27 O L (全 11 頁)

(21)出願番号 特願平9-272792

(22)出願日 平成9年(1997)10月6日

(71)出願人 000003078

株式会社東芝

神奈川県川崎市幸区堀川町72番地

(72)発明者 宮本 出

東京都青梅市末広町2丁目9番地 株式会
社東芝青梅工場内

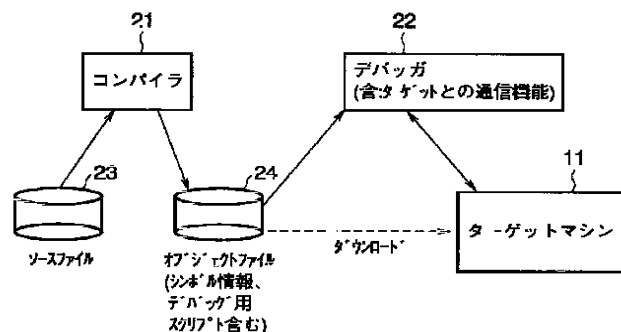
(74)代理人 弁理士 鈴江 武彦 (外6名)

(54)【発明の名称】 プログラムデバッグ装置、プログラムデバッグ方法及びその方法を記録したコンピュータ読取り可能な記録媒体

(57)【要約】

【課題】本発明はデバッグ時物件と最終物件と同じ実行オブジェクトで管理でき、デバッグ処理時にターゲットマシン側のメモリ負担がないようにオブジェクトファイルを生成しデバッグすることを課題とする。

【解決手段】ソースファイル23をもとにコンパイラ21により生成したオブジェクトファイル24は、デバッグ時起動のデバッグスクリプトファイルと、デバッグスクリプトを含まない実行オブジェクトファイルと、デバッグスクリプトとソース行の位置を対応付けるデバッグスクリプト起動情報のシンボルテーブルとに分けられる。ターゲットマシン11に実行オブジェクトファイルをダウンロードし、このファイルを実行しデバッグする際、デバッグスクリプト部分にプログラムの実行が到達した時、デバッグスクリプト起動情報に基づきデバugg22がホストマシン10上のデバッグスクリプトファイルを実行しデバッグを行う。



【特許請求の範囲】

【請求項1】 プログラムの実行時にデバッグ時にのみ実行する処理を記述するデバッグスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記デバッグスクリプト部分であるデバッグスクリプトファイルと、上記ソースファイルのソース行の位置と上記デバッグスクリプトを対応付けるデバッグスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルにコンパイルするコンパイラと、

デバッグ時に上記デバッグスクリプトを有効にするか否かを指定するデバッグ指定手段と、

このデバッグ指定手段により上記デバッグスクリプトを有効にしたとき、上記実行オブジェクトファイルを実行する際に上記デバッグスクリプトにプログラムの制御が到達したことを判定する判定手段とを具備し、

上記判定手段により上記デバッグスクリプトにプログラムの制御が到達したことを判定したときに、上記デバッグスクリプト起動情報に基づいて対応した上記デバッグスクリプトファイルを実行することを特徴とするプログラムデバッグ装置。

【請求項2】 上記コンパイラは、上記ソースファイルをコンパイルする際に、所定の記述により上記デバッグスクリプトを識別しコンパイルすることを特徴とする請求項1記載のプログラムデバッグ装置。

【請求項3】 上記ソースファイルは、上記デバッグスクリプトが他のプログラムと同一の言語により文法的に他のプログラム記述部と区別できる識別子で記述され、上記コンパイラは、上記ソースファイルをコンパイルする際に上記識別子にて上記デバッグスクリプトを識別しコンパイルすることを特徴とする請求項1記載のプログラムデバッグ装置。

【請求項4】 上記デバッグ指定手段は、上記デバッグスクリプトを有効にした場合、上記デバッグスクリプトを識別できる命令を上記デバッグスクリプト起動情報に基づき上記実行オブジェクトファイルの対応する位置に書き込み、

上記判定手段は、上記命令に基づき上記デバッグスクリプトにプログラムの制御が到達したことを判定することを特徴とする請求項1乃至3のいずれかに記載のプログラムデバッグ装置。

【請求項5】 プログラムの実行時にデバッグ時にのみ実行する処理を記述するデバッグスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記デバッグスクリプト部分であるデバッグスクリプトファイルと、上記ソースファイルのソース行の位置と上記デバッグスクリプトを対応づけるデバッグスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルに所定のステップ毎に翻訳し実行するインタプリタと、

デバッグ時に上記デバッグスクリプトを有効にするか否かを指定するデバッグ指定手段と、

このデバッグ指定手段により上記デバッグスクリプトを有効にした場合、上記実行オブジェクトファイルを実行する際に上記デバッグスクリプトにプログラムの制御が到達したことを判定する判定手段と、

この判定手段により上記デバッグスクリプトにプログラムの制御が到達したことを判定したときに、上記デバッグスクリプト起動情報に基づいて対応した上記デバッグスクリプトファイルを実行するプログラムデバッグ装置。

【請求項6】 上記インタプリタは、上記ソースファイルを翻訳する際所定の記述によりデバッグスクリプトを識別して翻訳し実行することを特徴とする請求項5記載のプログラムデバッグ装置。

【請求項7】 上記ソースファイルは、上記デバッグスクリプトが他のプログラムと同一の言語により文法的に他のプログラム記述部と区別できる識別子で記述され、上記インタプリタは、上記ソースファイルを翻訳する際に上記識別子にて上記デバッグスクリプトを識別して翻訳し実行することを特徴とする請求項5記載のプログラムデバッグ装置。

【請求項8】 上記デバッグ指定手段は、上記デバッグスクリプトを有効にした場合、上記デバッグスクリプトを識別できる命令を上記デバッグスクリプト起動情報に基づき上記実行オブジェクトファイルの対応する位置に書き込み、

上記判定手段は、上記命令に基づき上記デバッグスクリプトにプログラムの制御が到達したことを判定することを特徴とする請求項5乃至7のいずれかに記載のプログラムデバッグ装置。

【請求項9】 プログラムの実行時にログを出力するコードであるログスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記ログスクリプト部分であるログスクリプトファイルと、上記ソースファイルのソース行の位置と上記ログスクリプトを対応づけるログスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルにコンパイルするコンパイラと、

デバッグ時に上記ログスクリプトを有効にするか否かを指定するデバッグ指定手段と、

このデバッグ指定手段により上記ログスクリプトを有効にした場合、上記実行オブジェクトファイルを実行する際に上記ログスクリプトにプログラムの制御が到達したことを判定する判定手段と、

この判定手段により上記ログスクリプトにプログラムの制御が到達したことを判定した時に、上記ログスクリプト起動情報に基づいて対応した上記ログスクリプトファイルを実行するプログラムデバッグ装置。

【請求項10】 上記コンパイラは、上記ソースファイ

ルをコンパイルする際所定の記述によりログスクリプトを識別しコンパイルすることを特徴とする請求項9記載のプログラムデバッグ装置。

【請求項11】 上記ソースファイルは、上記ログスクリプトが他のプログラムと同一の言語により文法的に他のプログラム記述部と区別できる識別子で記述され、上記コンパイラは、上記ソースファイルをコンパイルする際に上記識別子にて上記ログスクリプトを識別しコンパイルすることを特徴とする請求項9記載のプログラムデバッグ装置。

【請求項12】 上記デバッグ指定手段は、上記ログスクリプトを有効にした場合、上記ログスクリプトを識別できる命令を上記ログスクリプト起動情報に基づき上記実行オブジェクトファイルの対応する位置に書き込み、上記判定手段は、上記命令に基づき上記ログスクリプトにプログラムの制御が到達したことを判定することを特徴とする請求項9乃至11のいずれかに記載のプログラムデバッグ装置。

【請求項13】 プログラムの実行時にデバッグ時にのみ実行する処理を記述するデバッグスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記デバッグスクリプト部分であるデバッグスクリプトファイルと、上記ソースファイルのソース行の位置と上記デバッグスクリプトを対応付けるデバッグスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルにコンパイルし、デバッグ時に、上記デバッグスクリプトを有効にするか否かを指定し、上記デバッグスクリプトを有効にした場合、上記実行オブジェクトファイルを実行する際に上記デバッグスクリプトにプログラムの制御が到達したことを判定し、上記デバッグスクリプトにプログラムの制御が到達したことを判定した際に、上記デバッグスクリプト起動情報に基づいて対応した上記デバッグスクリプトファイルを実行することを特徴とするプログラムデバッグ方法。

【請求項14】 上記ソースファイルをコンパイルする際に所定の記述により上記デバッグスクリプトを識別しコンパイルすることを特徴とする請求項13記載のプログラムデバッグ方法。

【請求項15】 上記ソースファイルは、上記デバッグスクリプトが他のプログラムと同一の言語により文法的に他のプログラム記述部と区別できる識別子で記述され、上記ソースファイルをコンパイルする際に上記識別子にて上記デバッグスクリプトを識別しコンパイルすることを特徴とする請求項13記載のプログラムデバッグ方法。

【請求項16】 上記デバッグスクリプトを有効にした場合、上記デバッグスクリプトを識別できる命令を上記デバッグスクリプト起動情報に基づき上記実行オブジェ

クトファイルの対応する位置に書き込み、

上記命令に基づき上記デバッグスクリプトにプログラムの制御が到達したことを判定することを特徴とする請求項13乃至15のいずれかに記載のプログラムデバッグ方法。

【請求項17】 プログラムの実行時にデバッグ時にのみ実行する処理を記述するデバッグスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記デバッグスクリプト部分であるデバッグスクリプトファイルと、上記ソースファイルのソース行の位置と上記デバッグスクリプトを対応付けるデバッグスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルに所定のステップ毎に翻訳し実行し、デバッグ時に上記デバッグスクリプトを有効にするか否かを指定し、

上記デバッグスクリプトを有効にした場合、上記実行オブジェクトファイルを実行する際に上記デバッグスクリプトにプログラムの制御が到達したことを判定し、上記デバッグスクリプトにプログラムの制御が到達したことを判定した際に、上記デバッグスクリプト起動情報に基づいて対応した上記デバッグスクリプトファイルを実行することを特徴とするプログラムデバッグ方法。

【請求項18】 上記ソースファイルを翻訳する際に所定の記述によりデバッグスクリプトを識別して翻訳し実行することを特徴とする請求項17記載のプログラムデバッグ方法。

【請求項19】 上記ソースファイルは、上記デバッグスクリプトが他のプログラムと同一の言語により文法的に他のプログラム記述部と区別できる識別子で記述され、

上記ソースファイルを翻訳する際に上記識別子にて上記デバッグスクリプトを識別して翻訳し実行することを特徴とする請求項17記載のプログラムデバッグ方法。

【請求項20】 上記デバッグスクリプトを有効にした場合、上記デバッグスクリプトを識別できる命令を上記デバッグスクリプト起動情報に基づき上記実行オブジェクトファイルの対応する位置に書き込み、上記命令に基づき上記デバッグスクリプトにプログラムの制御が到達したことを判定することを特徴とする請求項17乃至19のいずれかに記載のプログラムデバッグ方法。

【請求項21】 プログラムの実行時にログを出力するコードであるログスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記ログスクリプト部分であるログスクリプトファイルと、上記ソースファイルのソース行の位置と上記ログスクリプトを対応付けるログスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルにコンパイルし、

デバッグ時に上記ログスクリプトを有効にするか否かを指定し、

上記ログスクリプトを有効にした場合、上記実行オブジェクトファイルを実行する際に上記ログスクリプトにプログラムの制御が到達したことを判定し、

上記ログスクリプトにプログラムの制御が到達したことを判定した際に、上記ログスクリプト起動情報に基づいて対応した上記ログスクリプトファイルを実行するプログラムデバッグ方法。

【請求項22】 上記ソースファイルをコンパイルする際所定の記述によりログスクリプトを識別しコンパイルすることを特徴とする請求項21記載のプログラムデバッグ方法。

【請求項23】 上記ソースファイルは、上記ログスクリプトが他のプログラムと同一の言語により文法的に他のプログラム記述部と区別できる識別子で記述され、上記ソースファイルをコンパイルする際に上記識別子にて上記ログスクリプトを識別しコンパイルすることを特徴とする請求項21記載のプログラムデバッグ方法。

【請求項24】 上記ログスクリプトを有効にした場合、上記ログスクリプトを識別できる命令を上記ログスクリプト起動情報に基づき上記実行オブジェクトファイルの対応する位置に書き込み、上記命令に基づき上記ログスクリプトにプログラムの制御が到達したことを判定することを特徴とする請求項21乃至23のいずれかに記載のプログラムデバッグ方法。

【請求項25】 プログラムの実行時にデバッグ時のみ実行する処理を記述するデバッグスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記デバッグスクリプト部分であるデバッグスクリプトファイルと、上記ソースファイルのソース行の位置と上記デバッグスクリプトを対応付けるデバッグスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルにコンパイルし、デバッグ時に上記デバッグスクリプトを有効にするか否かを指定し、

上記デバッグスクリプトを有効にした場合、上記実行オブジェクトファイルを実行する際に上記デバッグスクリプトにプログラムの制御が到達したことを判定し、上記デバッグスクリプトにプログラムの制御が到達したことを判定した際に、上記デバッグスクリプト起動情報に基づいて対応した上記デバッグスクリプトファイルを実行するプログラムデバッグ方法のプログラム情報を格納したコンピュータ読み取り可能な記録媒体。

【請求項26】 プログラムの実行時にデバッグ時のみ実行する処理を記述するデバッグスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記デバッグスクリプト部分であるデバッグスクリプトファイルと、上記ソースファイ

ルのソース行の位置と上記デバッグスクリプトを対応付けるデバッグスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルに所定のステップ毎に翻訳し実行して、

デバッグ時に上記デバッグスクリプトを有効にするか否かを指定し、

上記デバッグスクリプトを有効にした場合、上記実行オブジェクトファイルを実行する際に上記デバッグスクリプトにプログラムの制御が到達したことを判定し、

上記デバッグスクリプトにプログラムの制御が到達したことを判定した際に、上記デバッグスクリプト起動情報に基づいて対応した上記デバッグスクリプトファイルを実行するプログラムデバッグ方法のプログラム情報を格納したコンピュータ読み取り可能な記録媒体。

【請求項27】 プログラムの実行時にログを出力するコードであるログスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記ログスクリプト部分であるログスクリプトファイルと、上記ソースファイルのソース行の位置と上記ログスクリプトを対応づけるログスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルにコンパイルし、

デバッグ時に上記ログスクリプトを有効にするか否かを指定し、

上記ログスクリプトを有効にした場合、上記実行オブジェクトファイルを実行する際に上記ログスクリプトにプログラムの制御が到達したことを判定し、

上記ログスクリプトにプログラムの制御が到達したことを判定した際に、上記ログスクリプト起動情報に基づいて対応した上記ログスクリプトファイルを実行するプログラムデバッグ方法のプログラム情報を格納したコンピュータ読み取り可能な記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、コンピュータプログラム開発時のコンパイラと、デバッグを備えたプログラムデバッグ装置、プログラムデバッグ方法、及びその方法を記録したコンピュータ読み取り可能な記録媒体に関する。

【0002】

【従来の技術】従来、組込みシステムのアプリケーション開発の際等に於いて、ターゲットシステム上のプログラムのデバッグを行なう際、エラーチェックルーチンを組込んでデバッグを行ない、プログラムをリリースする際にはエラーチェックルーチンを含まないコードを改めて生成するケースが多々あった。

【0003】この場合は、再コンパイルが必要であり開発効率が悪い。また、エラーチェックルーチンの有無というデバッグ時とリリース時ではプログラム自体が異なることによる物件管理面での問題、エラーチェックルー

チンを含むためデバッグ時により多くのメモリを必要とするという問題もあった。

【0004】一般に、プログラム開発者は、プログラムを記述する場合、それをデバッグすることを意識して作成する場合が多い。具体的には、ソースファイルにエラーチェックルーチンを含めておくことが多い。そしてコンパイルオプションでこれを有効にしてコンパイルし、実行モジュールを作成してデバッグする。

【0005】デバッグが完了すると、このようなコードを無効にしてコンパイルすることで最終プログラムを生成する。特に、組込み製品でのプログラムでは最終的にエラー処理ルーチンは除きたい場合が多々ある。

【0006】また、多くのデバッグでは、スクリプト記述が可能であり、新たなデバッグコマンドを記述したりすることはできるが、ソースプログラムと独立である。具体的なイメージを擬似言語で記述すれば、以下のようなコードを開発者が記述することになる。

```

【0007】
Function F () {
    . . . . .
    #ifdef DEBUG
        error check code . . .
    #endif
    . . . . .
    #ifdef DEBUG
        error check code . . .
    #endif
    . . . . .
}
. . . . .

```

DEBUGをオンにして（コンパイルして）オブジェクトを生成しデバッグ完了したあと、DEBUGをオフにして再度（コンパイルして）オブジェクトを生成し最終物件を作成する。

【0008】これでは、デバッグ時のオブジェクトと最終的なオブジェクトとが別ものになり、管理上の問題を生じたり、再コンパイルが必要で開発効率が悪い。また、デバッグ時では、最終プログラムより多くのコードを含むため、一時的にせよ、メモリを多く必要としてしまう。これは、メモリに厳しい制限のある組込み製品などの場合、問題となる場合がある。

【0009】

【発明が解決しようとする課題】上述したように問題点を解決するには、以下のような開発環境が提供されることが望ましい。1. デバッグ時物件と、最終物件とを同じ実行オブジェクトで管理できる。

【0010】2. デバッグ処理時に、ターゲットマシン側のメモリ負担がないようにする。これらの開発環境は、コンパイラやデバッグに係わる部分を従来と変更することで実現される。

【0011】そこで、本発明は上記事情を考慮してなされたもので、デバッグ用スクリプト記述やプログラムの記述言語による（主にエラーチェックなどの）記述を、ソースファイル内に記述可能で、コンパイラが自動的にそれを抽出してオブジェクトファイル内に保存し、デバッグ時の開発者（デバッグ操作者）の要求に従い、プログラム実行時に自動的にデバッグ用スクリプト記述やプログラムの記述言語による（主にエラーチェックなどの）記述を実行することで、デバッグ時物件と最終物件とを同一のオブジェクトファイルとして扱うことができ、且つクロス開発の場合に、デバッグのためのコードをターゲットマシン内にロードすることを回避避することができるプログラムデバッグ装置、及びプログラム管理方法、並びにその方法を記録したコンピュータ読取り可能な記録媒体を提供することを目的とする。

【0012】

【課題を解決するための手段】本発明は、上記目的を達成するため、本発明のプログラムデバッグ装置は、プログラムの実行時にデバッグ時にのみ実行する処理を記述するデバッグスクリプトを含んだソースファイルを、通常のプログラム部分である実行オブジェクトファイルと、上記デバッグスクリプト部分であるデバッグスクリプトファイルと、上記ソースファイルのソース行の位置と上記デバッグスクリプトを対応付けるデバッグスクリプト起動情報を持つシンボルテーブルとを含むオブジェクトファイルにコンパイルするコンパイラと、デバッグ時に上記デバッグスクリプトを有効にするか否かを指定するデバッグ指定手段と、このデバッグ指定手段により上記デバッグスクリプトを有効にした場合、上記実行オブジェクトファイルを実行する際上記デバッグスクリプトにプログラムの制御が到達したことを判定する判定手段と、この判定手段により上記デバッグスクリプトにプログラムの制御が到達したことを判定した時に、上記デバッグスクリプト起動情報に基づいて対応した上記デバッグスクリプトファイルを実行することを特徴とする。

【0013】この構成によれば、デバッグ時のプログラムと最終プログラムを同一のオブジェクトファイルとして扱うことができる。また、デバッグのためのコードは、別のオブジェクトファイルとして管理するので、デバッグのためのコードをターゲットマシン内にロードすることを避けることができる。さらに、上記構成のコンパイラをインタプリタに置き換えても同様な効果が得られる。そして、デバッグスクリプトをログ出力コードに置き換え、このログ出力コードをデバッグ時に利用しても同様な効果が得られる。

【0014】

【発明の実施の形態】以下、図面を参照しながら本発明の一実施の形態を説明する。

（第1の実施形態）図1は本発明の実施形態の前提となるシステム構成を示すブロック図である。

【0015】ここではデバッグ対象となるプログラムが動作するマシンをターゲットマシン11、プログラムを開発（生成）するマシンをホストマシン10と呼ぶ。両者は、同一のハードウェアであっても良いが、ここでは以下のようにターゲットマシン11で動作するプログラムを、別個のホストマシン10を用いて開発する場合を想定する。また、両者は特に図示しないが、CPU、メモリ、HDD等の補助記憶装置、表示装置、通信手段等、通常のコンピュータが備える構成を当然に有しているものとする。そして、両者はLAN（Local Area Network）やシリアル等の通信媒体／手段たる例えばネットワーク12で接続されており、互いにデータを送受信する。

【0016】図2を参照して、ホストマシン10上の構成を説明する。ホストマシン10では、プログラムソース（ソースファイル）23をコンパイラ21でコンパイルする。そして、作成されたオブジェクトファイル24より、実行イメージをホスト上のデバッガ22が、ネットワーク12を介して、ターゲットマシン11上にダウンロードする。

【0017】デバッガ22は、プログラムのシンボル情報を取り込むことで、ユーザはシンボリックなデバッグをホストマシン10上で行うことができる。ユーザの記述したソースファイル23は、通常のプログラム以外に「デバッグスクリプト」を含めることができる。これは、デバッグ時に、オプションにより該当する部分にプログラムが到達した場合、そのデバッグスクリプトを実行することを保証するものである。このオプションを指定しなければ、この部分は全く実行されない。

【0018】本ツールでは、コンパイラが通常のオブジェクトファイル24内に、デバッグスクリプトを保存するセクションを定義する。これにより、コンパイルしたオブジェクトファイル24には、プログラムコードに相当するセクション（テキストセクション）、データに相当するセクション（データセクション）等を含む通常のプログラムの実行オブジェクトファイルのセクションと、デバッグスクリプトのオブジェクトファイルを保存するデバッグスクリプトセクションと、デバッグスクリプトとソース行の位置を対応づけるデバッグスクリプト起動情報等を含むシンボルテーブルが含まれる。

【0019】デバッガ22は、デバッグスクリプトを利用しない場合、通常のリモートデバッグシステムと同様

```
func(int a, int b){
    int c;
#ifdef debug_by_command
    watch a
    print a
#endif
#ifdef debug_by_C
```

な様相を呈する。デバッグスクリプトオプションが指定された場合でも、プログラムのターゲットマシン11へのダウンロード機能については全く同じである。

【0020】図3を参照して、ターゲットマシン11の構成を説明する。本実施形態では、通常のターゲットシステムでよく用いられるターゲットモニタ31を仮定するが必須ではない。ターゲットモニタ31は、ホストマシン10との通信機能を備え、デバッグ実行時にブレーク命令により例外が発生した場合ホストマシン10に通知する。32は、ホストマシン10からダウンロードされた対象プログラム（オブジェクトファイル）である。

【0021】次に、ホスト側の言語処理システムの説明をする。ユーザプログラムは、通常の高級プログラミング言語と、この言語のコンパイラの存在を仮定する。上記コンパイラは、特に図示しないが、構文解析部、最適化部、コード生成部などから構成される。その内、本実施形態で主に関係があるのは、構文解析部である。

【0022】構文解析部では、デバッグスクリプト部とそうでない部分を切り分けて処理する。この識別は、プログラム開発者がソースファイルを作成する際に、「コンパイラ指示子」をソースファイル内に埋め込むことで行う。デバッグ用スクリプトの具体的実施例としては、「ソースプログラム本体と同じ言語（場合によっては、文法的にサブセット）によって記述」、もしくは「デバッガコマンドからなる記述」、もしくは上記が混在したものが考えられる。

【0023】C言語風の表現では、以下のようにソースファイルの記述の中に、コンパイル指示子で、デバッグコマンド部分などを記述する。コンパイラの構文解析部は、この部分を通常のプログラムと別の処理を行う。

【0024】下記例でコンパイル指示子とは、「#if def debug_by_command ... #endif」「#if def debug_by_C ... #endif」を指す。これらは、プログラムの本来の記述ではなく、コンパイルするときのコンパイラに対する動作指示となる。

【0025】この例では、コンパイラは、「#if def debug_by_command」を発見すると、ここからは普通のプログラムではなく、デバッグコマンドであると認識して処理する。また、同様に「#if def debug_by_C」を発見すると、ここからはデバッグコマンドに翻訳すると認識する。

```
//ここからデバッガコマンド
// デバッガコマンドの例
```

```
//C言語のチェック
// →デバッグコマンドに変換する
```

```

c= convert (a, b);
// →関数の一時実行コマンドに変換
// call $1 = check(a, b)
// set c = $1
if (a > b) printf ("error:a = %d\n ", a); // →式の評価に変換:
// set $2 = a
// set $3 = b
// if ($2 > $3) call printf
//("error:a = %d\n ", a)

#endif
c = ....
....
}

```

上記のように「コンパイラ指示子」では、その記述言語の種類を指定する。今、その種類として、ソースと同一のプログラム言語、デバッグコマンドを想定する。ソースと同一のプログラム言語の場合、これを自動的にデバッグコマンドに変換する。

【0026】図4を参照して、ホストマシン10上での上記の様なソースファイルのコンパイル処理の流れを説明する。ソースファイル23をコンパイラ21にてコンパイルする場合、まず、コンパイラ21はコンパイラ指示子が言語記述か否かを判定する(ステップA1)。

【0027】言語記述であれば(ステップA1のyes)、デバッグスクリプトへの変換ルーチン呼び出し(ステップA4)、デバッグスクリプトセクションを定義して翻訳する(ステップA5)。もし、デバッグスクリプトへ変換できなかった場合はエラー処理(ステップA6)として終了する。

【0028】言語記述でなければ(ステップA1のno)、次にコンパイラ指示子がデバッグコマンドか否かを判定する(ステップA2)。コンパイラ指示子がデバッグコマンドである場合(ステップA2のyes)、デバッグスクリプトセクションを定義して翻訳する(ステップA5)。コンパイラ指示子がデバッグコマンドでない場合(ステップA2のno)、従来の処理と同様となる(ステップA3)。

【0029】図5に、ホストマシン10上でのソースファイル23のコンパイル処理によるオブジェクトファイル生成の概念図を示す。ソースファイル23の一例であるソースファイル50をコンパイラ21により、コンパイルして生成されるオブジェクトファイルは、実行オブジェクトファイル51、デバッグスクリプトファイル52、シンボルテーブル53の3つの部分に分けられる。

【0030】実行オブジェクトファイル51は、デバッグスクリプト部を含まない実行オブジェクトファイルのセクションである。デバッグスクリプトファイル52は、デバッグスクリプトのオブジェクトファイルであるデバッグ用のスクリプトファイルセクションであり、デバッグ時に使用される。シンボルテーブル53は、シンボル情報を含むファイルのセクションである。シンボル

情報とは、例えば、デバッグ用スクリプトを起動する際に用いるソース行とそれに対応する実行オブジェクトファイルのロケーション(位置)とのマッピング情報等である。

【0031】そして、オブジェクトファイル51である当該セクションのみがターゲットマシン11にロードされる。次に、ホストマシン10上でのデバッグ時の動作説明をする。

【0032】まず、図6により、通常のブレークポイント指定処理の流れを説明する。ブレークポイントとは、デバッグ時にデバッグ操作者が、必要に応じて実行オブジェクトファイルの実行を中断するためのポイントであり、このブレークポイントを示すため上記実行オブジェクトファイルにはブレーク命令が挿入される。また、この例でのデバッグは、上記ソースファイル50をコンパイルして生成された実行オブジェクトファイル51について行うこととし、その際、デバッグスクリプトファイル52、シンボルテーブル53を参照する。

【0033】ブレークポイントの指定は、デバッグ操作者がブレークポイントに対応するソース行を指定して行う(ステップB1)。ホストのデバッグ22が、シンボルテーブル53にあるシンボル情報(ここでは、ソース行とそれに対応する実行オブジェクトファイル51のロケーションとのマッピング情報)に基づき、上記指定したブレークポイントに対する実行オブジェクトファイル51のロケーションにブレーク命令を書き込む(ステップB2)。その際に、上記ステップB1で指定したブレーク命令を書き込んだ実行オブジェクトファイル51のロケーションに有った元々の命令はデバッグ22のメモリの中に保存しておく(ステップB3)。

【0034】次に、図7により、通常のブレークポイントの解除処理の流れを説明する。デバッグ操作者が、ブレークポイントを解除する(ステップC1)。デバッグ22は、実行オブジェクトファイル51の解除されたブレークポイントのブレーク命令のロケーションに、メモリに保存しておいたこのブレーク命令のロケーションに有った元々の命令を書き込む(ステップC2)。これにより、このロケーションの命令を実行してもブレーク命

令による例外は発生しない。

【0035】本実施形態では、デバッグ時にターゲットマシン11で実行オブジェクトファイル51を実行する際に、実行オブジェクトファイル51に挿入したブレーク命令にプログラムの処理が到達した時、このブレーク命令がデバッグスクリプトに対応している場合、ホストマシン10上のシンボルテーブル53のシンボル情報に基づき、ホストマシン10上のデバッグスクリプトファイル52からこのブレークポイントに対応したデバッグスクリプトを実行して、デバッグを行う。

【0036】デバッグ操作者によるデバッグスクリプト有効化または無効化の指示について以下に説明する。まず、図8により、デバッグスクリプト有効化処理の流れを説明する。デバッグ操作者が、デバッグスクリプト有効化を指定する(ステップD1)。デバッグスクリプト有効化が指定されると、デバッグ22は、シンボルテーブル53より、各デバッグスクリプト毎に、実行オブジェクトファイル51の該当するロケーションを調べそこにブレーク命令を書き込む(ステップD2)。ブレーク命令のパラメータとして、各デバッグスクリプト毎のIDを示すデバッグスクリプトIDを挿入しておく。ステップD2の際に、実行オブジェクトファイル51の該当するロケーションにあった元々の命令をデバッグのメモリに保存しておく(ステップD3)。

【0037】続いて図9により、デバッグスクリプト無効化処理の流れを説明する。デバッグ操作者が、デバッグスクリプト無効化を指定する(ステップE1)。デバッグスクリプト無効化が指定されると、デバッグ22は、シンボルテーブル53より、各デバッグスクリプト毎に、実行オブジェクトファイル51の該当するロケーションを調べ、対応するブレーク命令のロケーションにデバッグの保存領域にコピーしてあったそのロケーションに有った元々の命令を書き込む(ステップE2)。これにより、デバッグ時に、ブレーク命令によるブレーク例外は発生せずデバッグスクリプトは無視される。

【0038】図10により、デバッグ時の処理の流れを説明する。デバッグ時、ターゲットマシン11で実行オブジェクトファイル51を実行する際、ブレーク命令にプログラムが到達すると(ステップF1)と、通常のCPUでは例外が発生する。例外とは、外部からの割り込み、ゼロによる割り算などのときにも起こるもので、これが発生するとCPUは、現在のプログラムの実行を止めて、図示しない例外処理ハンドラにジャンプする。

【0039】例外処理ハンドラでは、どのような例外が発生してここに飛んできたか、CPUのレジスタなどを

参照して判定し、それぞれサブの処理ルーチンに飛ばす。例外が発生した時、それが「ブレーク命令」によって起こったものとわかると、ターゲットモニタ31にそれを知らせる。ターゲットモニタ31は、内蔵する通信機能によりホストマシン10のデバッグ22に、ブレークが起きたことを知らせる。

【0040】ホストマシン10のデバッグ22は、ブレークポイントを張っていることを分かっているため、デバッグ操作者にそれをわかるようにターゲットマシン11の表示装置に表示する。

【0041】ターゲットモニタ31は、次のデバッグ22からのコマンドを待つ。ここでは、まずホストマシン10上のデバッグ22は、発生した例外がデバッグ例外か否かを判定する(ステップF2)。

【0042】デバッグ例外でない場合(ステップF2のno)、通常の例外処理を実行し(ステップF6)、終了する。デバッグ例外で有る場合(ステップF2のyes)、デバッグスクリプト対応か否かを判定する(ステップF3)。

【0043】デバッグスクリプト対応でない場合(ステップF3のno)、通常のデバッグ例外処理を実行し(ステップF7)、終了する。デバッグスクリプト対応である場合(ステップF3のyes)、デバッグ22は、デバッグスクリプトIDを調べ、シンボルテーブル53のシンボル情報に基づきホストマシン10上のデバッグスクリプトファイル52から各デバッグコマンドを実行し(ステップF4)、例外処理より復帰する(ステップF5)。例外処理より復帰したら、続けてプログラムを実行する。

【0044】以上のようにすることで、デバッグ時プログラムと最終プログラムを同一のオブジェクトファイルとして扱うことができ、またデバッグのためのコードをターゲットマシン内にロードすることを避けることができる。

【0045】上述の第1の実施形態では、コンパイラを仮定していたが、インタプリタ方式でも同様に扱うことができる。また、プログラム言語としては、複数種の存在を仮定することもできる。さらに、コンパイル指示子の種類により、記述の種類を判別したが、より高度にコンパイラでの自動判定することが考えられる。

【0046】所定の記述、例えば、関数名、変数名がdebug_...を含む式や、それを含む文を判定しデバッグコマンドに自動変換するなど。例として、以下の様なものがあげられる。

```
func (int a, int b){
    int c;

    debug_watch a           //デバッグコマンドとして処理: watch a
    debug_print a           //デバッグコマンドとして処理: print a
```

```

printf ("%d\n", c);
c = debug_convert (a, b);

}

//通常プログラムとして処理
//デバッグコマンドとして処理：
// call $1 = check(a, b)
// set c = $1

```

(第2の実施形態)次に、本発明に係わる第2の実施形態について説明する。

【0047】本実施形態は、上記第1の実施形態でデバッグスクリプトに相当する部分を、デバッグ時にログとして出力されるコードに置き換えたものと考えることができる。

【0048】本実施形態でのソースファイル例であるソースファイル70をコンパイルしてオブジェクトファイル生成する概念図を図11に示す。ソースファイル70をコンパイラにてコンパイルすると、生成されるオブジェクトファイルは、実行オブジェクトファイル71、ログスクリプトファイルセクション72、シンボルテーブル73の3つの部分に分けられる。

【0049】実行オブジェクトファイル71は、デバッグスクリプト部(この場合ログとして出力されるコード)を含まない実行オブジェクトファイルのセクションである。ログスクリプトファイルセクション72は、デバッグ用のスクリプトファイルセクションであり、ログ出力がデバッグ用に使用される。

【0050】シンボルテーブル73は、シンボル情報を含むファイルのセクションである。シンボル情報とは、例えばデバッグ用スクリプトであるログを起動する際に用いるソース行とそれに対応する実行オブジェクトファイルのロケーションとのマッピング情報等である。

【0051】本実施形態のログ出力の有効化、無効化処理は、第1の実施形態のデバッグスクリプトの有効化、無効化処理と同様に扱うことができる。ログ出力が有効化されれば、ログ出力命令のブレークポイントにプログラムが到達した場合、ログスクリプトファイルセクション72からこのログ出力命令に対応するログ出力を選択して実行しデバッグを行うことができる。

【0052】

【発明の効果】以上詳記したように本発明によれば、デバッグ時物件と最終物件とを同一のオブジェクトファイルとして扱うことができ、デバッグのためのコードをターゲットマシン内にロードすることを回避することで、ターゲットマシンのメモリ要求量を抑制することができる。また、ソースファイルとデバッグ用スクリプトを一括して管理できる。そして、デバッグスクリプトとし

て、プログラムを記述している言語そのものが使用でき、プログラマ(プログラム開発者)からみると、それがデバッグコマンドに翻訳されて実行されるということ意識しないで良いという優れた効果を奏する。

【図面の簡単な説明】

【図1】本発明の第1の実施形態に於ける装置の概略構成を示す図。

【図2】同実施の形態に係るホストマシン上の構成要素を示す概念図。

【図3】同実施の形態に係るターゲットマシン上の構成要素を示す概念図。

【図4】同実施の形態に係るソースファイルのコンパイル処理の流れを示すフローチャート。

【図5】同実施の形態に係るソースファイルをコンパイルして生成されるオブジェクトファイルの概念図。

【図6】同実施の形態に係る通常のブレークポイント指定処理の流れを示すフローチャート。

【図7】同実施の形態に係る通常のブレークポイント解除処理の流れを示すフローチャート。

【図8】同実施の形態に係るデバッグスクリプト有効化処理の流れを示すフローチャート。

【図9】同実施の形態に係るデバッグスクリプト無効化処理の流れを示すフローチャート。

【図10】同実施の形態に係るデバッグ時の処理の流れを示すフローチャート。

【図11】本発明の第2の実施形態に於ける、ソースファイルをコンパイルして生成されるオブジェクトファイルの概念図。

【符号の説明】

10…ホストマシン

11…ターゲットマシン

12…ネットワーク

21…コンパイラ

22…デバッグ

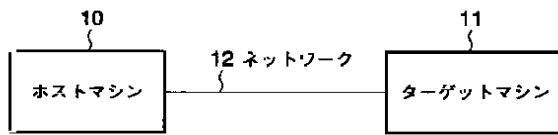
23…ソースファイル

24…オブジェクトファイル

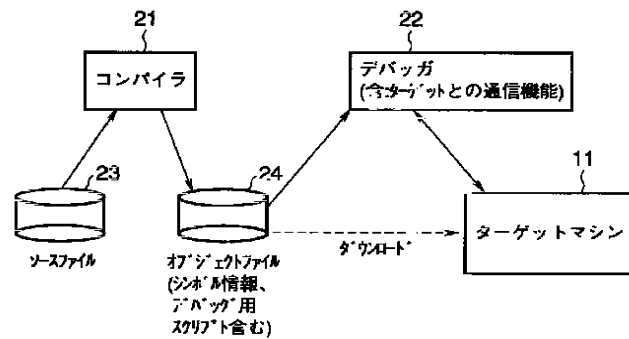
31…ターゲットモニタ

32…ダウンロードされた対象プログラム

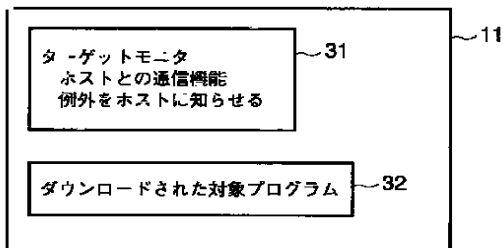
【図1】



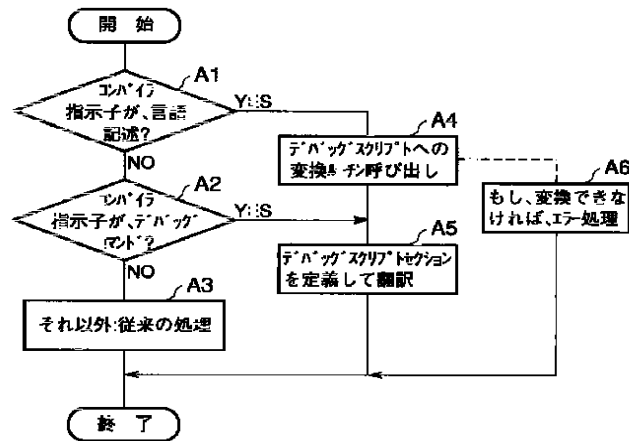
【図2】



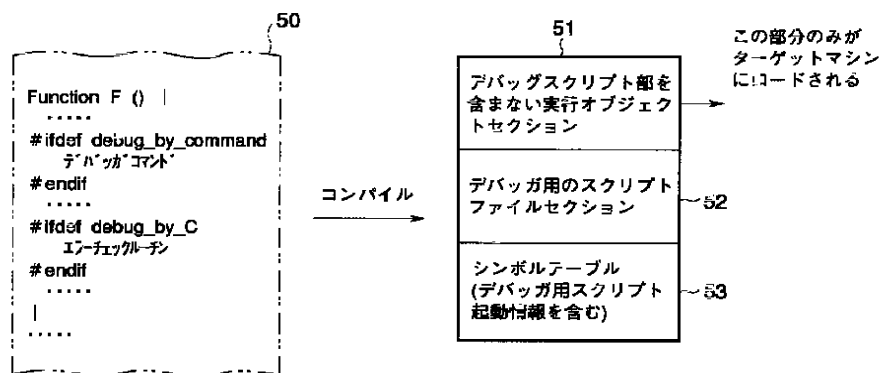
【図3】



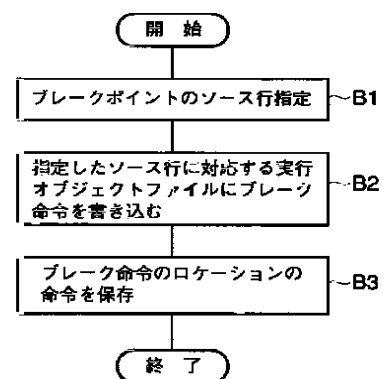
【図4】



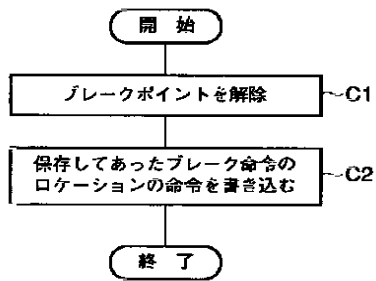
【図5】



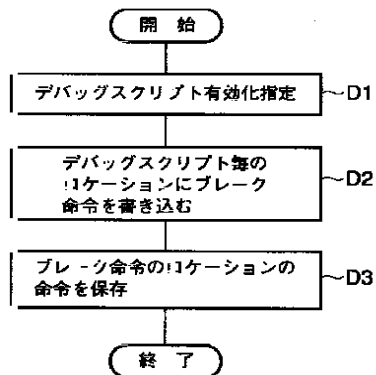
【図6】



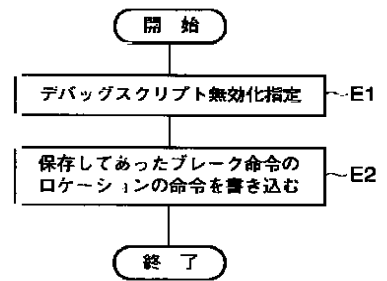
【図7】



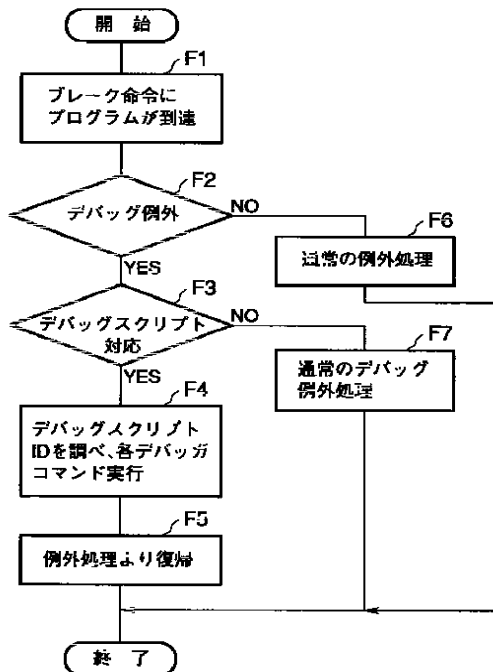
【図8】



【図9】



【図10】



【図11】

